1. itertools.permutations(iterable[, r]) : This tool returns successive  length permutations of elements in an iterable. If  is not specified or is None, then  defaults to the length of the iterable, and all possible full length permutations are generated.Permutations are printed in a lexicographic sorted order. So, if the input iterable is sorted, the permutation tuples will be produced in a sorted order.
2. **from** itertools **import** permutations
3. S, k = **input**().split()
4. **print**('\n'.join(**sorted**(**map**(''.join,permutations(S,**int**(k))))))

map() function returns a map object(which is an iterator) of the results after applying the given function to each item of a given iterable (list, tuple etc.) Syntax :

map(fun, iter)

The map() function is used to apply this function to each element of the numbers list, and an if statement is used within the function to perform the necessary conditional logic.

* # Return double of n
* def addition(n):
* return n + n
* # We double all numbers using map()
* numbers = (1, 2, 3, 4)
* result = map(addition, numbers)
* print(list(result))

2. itertools.combinations(iterable, r)

This tool returns the length subsequences of elements from the input iterable. Combinations are emitted in lexicographic sorted order. So, if the input iterable is sorted, the combination tuples will be produced in sorted order.

* **from** itertools **import** combinations
* S, k  = **input**().split()
* **for** i **in** **range**(1, **int**(k)+1):
* comb = **list**(combinations(**sorted**(S),i))
* **for** each **in** comb:
* **print**("".join(each))

3. Reverse an array of integers.

* **def** reverseArray(a):
* *# method 1*
* **return** a[::-1]
* # method 2
  + rev = []
* **for** i **in** **range**(**len**(a)):
* rev.append(a[**len**(a)-1-i])
* **return** rev
* **if** **\_\_name\_\_** == '\_\_main\_\_':
* fptr = **open**(os.environ['OUTPUT\_PATH'], 'w')
* arr\_count = **int**(**input**().strip())
* arr = **list**(**map**(**int**, **input**().rstrip().split()))
* res = reverseArray(arr)
* fptr.write(' '.join(**map**(**str**, res)))
* fptr.write('\n')
* fptr.close()

LeetCode

4. Roman to interger

* class Solution:
* def romanToInt(self, s: str) -> int:
* roman\_to\_integer = {
* 'I': 1,
* 'V': 5,
* 'X': 10,
* 'L': 50,
* 'C': 100,
* 'D': 500,
* 'M': 1000,
* }
* num = 0
* s = s.replace("IV", "IIII").replace("IX", "VIIII").replace("XL", "XXXX").replace("XC", "LXXXX").replace("CD", "CCCC").replace("CM", "DCCCC")
* #return sum(map(lambda x: roman\_to\_integer[x], s))
* for i in s:
* num += roman\_to\_integer[i]
* return num

1. Your task is to find out if the string contains: alphanumeric characters, alphabetical characters, digits, lowercase and uppercase characters.

**if** **\_\_name\_\_** == '\_\_main\_\_':

    s = **input**()

    methods = ["isalnum", "isalpha", "isdigit", "islower", "isupper"]

**for** i **in** methods:

        case = **False**

**for** k **in** s:

**if** **getattr**(k, i)():

                case = **True**

**break**

**print**(case)

Python getattr() function is used to access the attribute value of an object and also gives an option of executing the default value in case of unavailability of the key.

Python any() Function: The any() function returns True if any item in an iterable are true, otherwise it returns False. If the iterable object is empty, the any() function will return False.

* print (any(s.isalnum()for s in s))
* print (any(s.isalpha()for s in s))
* print (any(s.isdigit()for s in s))
* print (any(s.islower()for s in s))
* print (any(s.isupper()for s in s))
* return